Computer Science 301
Assignment Two

Steven Kerr 6022796

20/05/2019

Q1.
I would be sceptical until the proof is fully understood. Firstly, i would look at the definitions i knew. For instance, The Euclidean Algorithm is know to be ' a technique for quickly finding the GCD of two numbers'. The correctness proof of the Euclidean algorithm can be found on the internet. It involves proof by induction and there by proving for all integers.
In this sense, i know that the algorithm is correct. The 'Program correctness' proof would need more defining and investigation.
To start, I would define the meaning of 'Program correctness' in the following way:

1. there must be a formal specification for the program
2. there must be some formal definition of the semantics of the programming language

Such a definition may take the form of a set of axioms to cover the semantics of any simple statement in the language, and a set of inference rules that show how the semantics of any compound statement, including a complete program, can be inferred from the semantics of its individual component statements.
I also know from lectures that 'It makes sense to prove the correctness of an algorithm, but not, the correctness of a program.' This is, as i understand it, because programs are parallel to mathematical models. Even so, if the program is proven to be 'correct' there are still other variables that play a part in the program. As quoted from Donald Knuth: "Beware of bugs in the above code: I have only proved it correct, not tried it.".
In short, Proving program correctness is hard. The proofs often end up being long and tedious proof-by-cases. You need a formal semantics for whatever

language your program is specified in. For real world programming languages, developing such a semantics is often difficult, since there are possible corner cases or odd compiler behaviours that are difficult to model. Proving a program correct only proves that it matches a given specification. It also assumes that it's being compiled by a correct compiler, or run by a correct interpreter, which, in itself is another proof entirely.
If my class mate had gone to such efforts to satisfy each and every one of these requirements for 'Program Correctness' I would congratulate them on the amount of effort it would have taken and definitely consider trying to understand their proof.

Q2.
Firstly, I need to understand what W. P. Thurston is saying in his statement. I will break down and interpret his statement in the following way:
Thurston starts by outlining the idea that 'The standard of correctness and completeness' for a program is much higher than 'the mathematical community's standard of valid proofs'.
Thurston goes on to talk about the difficulties involved in writing a program. Even when coming close to the intellectual perspective of a decent mathematical paper, regardless of the amount of time and effort put in, will only ever be 'almost' formally correct.
Being that this constraint has been placed on programs Thurston argues that mathematics should also be considered 'almost' formally correct as well.

To begin with, I feel it is important for me to share my belief that mathematics and science(in relation to computer programming) are different. I also feel, therefore, that they should fit into separate classes of 'completeness and correctness'. It is very clear that mathematics is invaluable to science and programming, but to compare the study of abstraction(mathematics) and the physical world (programming) is hard. Better understood from a quote by Einstein: "As far as the laws of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality."

Programs are the implementation of logic (a collection of instructions) to facilitate specified computing operations and functionality. There are many physical factors involved in writing a program. Theses include:

- The choice of language to use. i.e Java, C, C++, Python... ect
- The computer being used. Can it handle the program requirements?. will the computer itself throw errors?

- Is the compiler correctly translating our list of commands into machine code?
- Are there 'bugs' in the code?, have the variables all been mapped correctly?

As we can see above, your implementation of the solution to any given problem or task is as important as the solution itself. This makes it hard when proving 'correctness of a program'.

Mathematics however is a completely different game. Mathematics is based on foundations, known as axioms, from which the rest of the subject is built from. Unlike in programming, the axioms of mathematics are unchanging.
Science can be seen as working in the opposite direction as mathematics. That is, determining the principles from the results, which is much harder than determining the results from the principles (mathematics).
For example, take the statement "there are infinitely many prime numbers." How can we know this to be really true? Well, we have a definition of the natural numbers through a set of axioms, and we have a definition of what it means to be a prime number. From those axioms we can logically derive that there are infinitely many primes. But that statement is implicitly conditioned on the axioms: We have to assume that what we are looking at really fulfils the Peano axioms. If we look at something which doesn't, the claim doesn't hold. However, mathematics doesn't look at a specific system. The statement it derives is not "for this real world object we have infinitely many primes." It says "Whenever we have something which fulfils those axioms, we know that we will find infinitely many primes." It also tells you that if we make certain other assumptions (such as that the axioms of set theory hold), we can derive that we'll find something fulfilling those axioms.

In conclusion, the level of 'correctness' needed for a program is definitely far higher than a mathematical proof, but this is in direct relation to a 'correctness' system designed for 'abstract' computation using a system of axioms. This system does not take into consideration the physicality involved with language choice, computer choice and compiling the instructions into machine code. Therefore, I do not feel that 'it is preposterous to claim that mathematics as we practice it is anywhere near formally correct.' but more that it is preposterous to assume we can prove correctness of a program using rules designed for mathematics.

Q3. a

Suppose a party has six people. Consider any two of them. They might be meeting for the first time, in which case we will call them mutual strangers; or they might have met before, in which case we will call them mutual acquaintances. The theorem says: In any party of six people either:

- at least three of them are (pairwise) mutual strangers or
- at least three of them are (pairwise) mutual acquaintances.

Q3. b

Suppose a graph has 6 vertices and every pair of vertices is joined by an edge. This is a complete graph. A complete graph on n vertices is denoted by the symbol K_n. K_6 has 15 edges and 6 vertices. Let the vertices represent 6 people at a party and let the edges be coloured red or blue depending on whether the two people represented by the vertices connected by the edge are mutual strangers or mutual acquaintances. No matter how you color the 15 edges you cannot avoid having either a red triangle, that is, a triangle all of whose three sides are red, representing three pairs of mutual strangers, or a blue triangle, representing three pairs of mutual acquaintances. In other words, whatever colours you use, there will always be at least two monochromatic triangle. If we try to create a K_5 we can see that the same idea does not hold. K_5 is a pentagon surrounding a star. If the star is all blue and the pentagon is all red then the theorem does not hold.